Fodis: a tool for Time-Constrained Scheduling

Marco Cassia
Ørsted
Technical University of Denmark
DK-2800 Kgs. Lyngby
Email:mca@oersted.dtu.dk

Abstract—Scheduling is a fundamental and perhaps the most important task in high-level synthesis. The NP complete nature of the scheduling problem makes necessary the use of heuristic algorithms. Fodis is a new tool implementing the Force-directed scheduling algorithm. The main advantage of the tool is its flexibility being the implementation fully based on dynamic data structure. The performance of Fodis is analyzed using classical benchmark; particular attention has been placed on the limits of the tool, which turn to be the limits of the FDS algorithm.

I. INTRODUCTION

Automated high-level synthesis of digital systems is a main research topic. Starting from an abstract behavioral description of a task, the goal of synthesis is to produce a complete datapath. The three central steps in the synthesis process are:

- Scheduling: assigning each operation to a specific control step.
- Allocation: determining the number and the type of functional units to execute the task.
- Binding: assigning operations to units and interconnecting components to form a complete datapath.

This paper is focused on the scheduling aspect of the synthesis process. Different methods based on different constraints (time constrains, resource constrains, time/resource constrains) are available to solve the scheduling problem. This paper discusses a time-constrained approach based on the Force-Directed Scheduling algorithm (FDS). Time-constrained scheduling is fundamental for real-time system designs. The paper is organized as follows: in section 2 the FDS algorithm is presented and analyzed. Section 3 discuss the implementation aspects of Fodis, a tool implementing FDS. In section 4 some scheduling example are shown and discussed. Section 5 presents some final considerations and possible future investigations.

II. FORCE-DIRECTED SCHEDULING ALGORITHM

The force-directed scheduling (FDS) algorithm [1] is a popular heuristic algorithm solving the time-constrained scheduling problem. Given an input Data Flow Graph (DFG) describing the task and the maximum number of steps to execute it, the algorithm assigns each operation to a specific control step trying to uniformly distributing the operations of each type. In this way the total number of functionals units required to execute the task is minimized.

It is possible to distinguish three main steps in the assignment of an operation to a specific control step:

- First the time frame of each operation is calculated.
- Based on the time frame, the operation distribution graph is built.
- Forces associated with each feasible assignment are calculated and the operation with the lowest force is scheduled.

Operations are mapped to nodes in the input DFG and topological constraints are represented by edges connecting the nodes. Each node can be moved within its time frame without violating time constraints. In order to determine the mobility of an operation, the As Soon As Possible (ASAP) algorithm and the As Late As Possible (ALAP) algorithm are run on the input DFG. The ASAP algorithm provides the earliest feasible control step for each operation; the ALAP algorithm provides the latest one. Moreover, the information provided by the ASAP algorithm permits to determine the minimum feasible number of control steps to execute the input task.

It is now possible to build the operation Distribution Graph (DG). The DG values are a measure of the distribution of operations of each type along the schedule interval of the task. The FDS algorithm tries to displace nodes in different control steps to make the distribution uniform. Formulas and examples for DG calculation can be found in [1].

The next step of the algorithm is to parse the input graph and to calculate the force for each feasible assignment of each node. What is the meaning of force? A mechanical analogy can help clarifying this concept: the force associated to a spring is given by: $F = kx$, where $k$ is a constant and $x$ is the spring displacement. The force associated to an operation assignment can be expressed in the same way: it is equal to a constant (the distribution of the operation) times the displacement (change in probability). The force is a measure to determine if the move is improving the DG balancing for that operation.

In the force calculation of an assignment both the direct force and the indirected forces need to be taken into account [1]. A change in the control step of a node may result into a new control step for the parents or the children nodes. This in turn can determine a different control step for the ancestors or successors of the parents/children nodes. The propagation effect of the initial move is quantified as an indirect force. The total force associated to a move is then given by the sum of the initial direct force (DF) plus the indirect forces (IF) [1].

The total force calculation results into a negative (better balancing) or a positive value (worse balancing). A total force equal to zero means that the operation distribution is not affected by the move. The force associated to nodes with a mobility equal to unity is obviously zero.

Finally, when all the nodes are visited, the operation asso-
Algorithm 1 Force directed scheduling algorithm

Collect graph information
Initialize data structure
Determine ASAP/ALAP c-steps for each node
Create Distribution Graph
SCHEDULE (GRAPH_NODES)
  REPEAT
    Set current_force = \infty
    FOR each unscheduled node n
      FOR every feasible step k
        Calculate the move force
        IF move_force < current_force
          current_force=move_force
          s_node=n
          s_step=k
        END IF
      END FOR
    END FOR
    Schedule s_node in s_step
    Update the nodes time-frame
    Update the Distribution Graph
  UNTIL all nodes are scheduled
Return final scheduling

III. Fodis

Fodis is a tool created to solve the scheduling of an input task using the FDS algorithm. It’s built in c code and compiled with the HLS libraries [2]. The HLS libraries provide data structures commonly used in high-level synthesis systems. Fodis is using the HLS graph and list structures together with the functions to handle them. An input file containing the Data Flow Graph of the task and the maximum number of control steps available for the scheduling are required as inputs. Once the execution is terminated, nodes with the assigned control step are printed on the screen. Fodis stores all the available information (e.g. DG table, DF, IF) during its execution in a log file (graph_out.log). This file is extremely useful for debugging.

The use of the HLS routines forces Fodis to accept DFG exclusively in the “.graph” format. Since such format is not directly readable by the xvcg [3] tool, Fodis produces two more output files: graph_in.vcg (the input DFG) and graph_out.vcg (the output DFG). These files can be directly read and displayed by the xvcg tool. Besides, since accepting exclusively files in the “.graph” format is a huge limitation, a Perl top level tool has been built around Fodis. The complete tool is able to handle also the “.dot” format input file; moreover, after Fodis termination, the tool displays directly the output scheduled datapath. For more detailed explanation please refer to the Readme file of Fodis.

All the previous features have been added in attempt to create a very flexible scheduling tool. For this reason a lot of efforts have been also put into the coding style. Fodis is a program exclusively built with dynamic data structures. Among the different implication (e.g. better memory utilization), the advantages of such coding style are:

- No limit to input nodes number.
- No limit to operation number and type (Fodis accepts all the strings).
- No limit to the schedule length of the task

The steps of the implementation of the top-level routine (SCHEDULE) are presented in fig. 1.

A. Initialization

The core of the initialization process is the GET_INFO routine. This function creates a data structure with multiple fields for each node of the DFG. These fields contain the information of the node, such as the ASAP/ALAP control steps and the type of operation associated to the node. The ASAP step of the starting nodes (no input edges) is initialized to control step 1. On the contrary, the ALAP step of the ending nodes (no output edges) is not initialized to the input schedule length provided by the user. The reason for this is due to the fact that the number inserted can be smaller than the minimum number of feasible steps (to be determined with the ASAP algorithm) to complete the task.

The next step of the initialization is the calculation of the time frame of each node through the ASAP and ALAP algorithms. These two algorithms follow the same implementation scheme; the ASAP algorithm is presented in fig. 2. After a node is selected, if the node ASAP (ALAP) cstep is not set, the ASAP (ALAP) scheduling of all the predecessors (successors) is controlled. If all the predecessors (successors) nodes are scheduled, then the ASAP (ALAP) cstep can be assigned to the selected node. The ASAP (ALAP) cstep is given by the max (min) ASAP (ALAP) cstep associated to the parents (children) of the selected node plus one (minus one), indicating that the delay of each operation is equal to one control step. The operation is repeated until all nodes are scheduled. Before proceeding further note that at this point it would be very easy to employ multi-cycle operations. It is just sufficient to replace the delay (here equal to one control step) with the execution time value associated to the operation.

Algorithm 2 ASAP algorithm

ASAP (GRAPH_NODES)
  REPEAT
    FOR each node n not scheduled
      IF all the ancestors n_anc of n are scheduled
        Set n_schedule_step=max(n_anc schedule_step)+1
      END IF
    END FOR
  UNTIL all nodes are scheduled
Return
B. Distribution graph

The DG is of fundamental importance in the forces calculation. The DG is implemented with a dynamic two-dimensional array. A row is associated to each control step; each column represents a certain operation type. The field opid in the data structure associated to the node represents the column entry of the DG table. The DG needs to be refreshed whenever a node is scheduled to reflect the changes of the scheduling in the operation distribution.

C. Force calculation

The algorithm for the force calculation MIN_FORCE is shown in fig. 3. The force is initially set to an arbitrary high value, which represents the boundary between a feasible and an unfeasible move. In the MIN_FORCE routine it is possible to distinguish 3 main steps executed for each unscheduled node and its respective moves:

- Calculate the direct force. This task is executed by the DIRECT_FORCE(NODE) routine.
- Calculate the total successors indirect forces. This task is execute by the SUCC_FORCE function. The main steps of the algorithm are presented in fig. 4. As shown the algorithm can be built in a very efficient way through recursion. For every child of the source node the direct force of the move (if any) is calculated. The resulting value is added to the current force value together with the force provided by the SUCC_FORCE(CHILD).
- Calculate the total ancestors indirect forces. This task is executed by the PREC_FORCE(NODE) function. The implementation is the same as the SUCC_FORCE(NODE), the only difference being the calculation made on each parent of the source node.

Once the total force is calculated, it is compared with the current force value. A lower value determines the annotation of the node and of the step into temporary variables, namely curr_node and t_schedule. When the loops terminate, curr_node points to the node with the lowest force between the unscheduled nodes and t_schedule contains the cstep at which the node needs to be scheduled. Note that the validity of the move of the successors/ancestors in the routines for the indirect force calculation is always guaranteed recursively.

Algorithm 3 MIN_FORCE routine.

MIN_FORCE (GRAPH_NODES)
Set current_force = \infty
FOR each unscheduled node n
FOR every feasible step k of node n
Calculate the Direct Force for n assigned to k
Calculate the successors Indirect Forces
Calculate the ancestors Indirect Forces
TF=DF+IP_succ+IF_anc
IF (TF < current_force)
    t_schedule=k
    curr_node=n
    current_force=TF
END IF
END For
END For
Return (curr_node,t_schedule)

D. Node scheduling and information updating

MIN_FORCE returns the node to be scheduled. Scheduling requires the updating of the fields of the data structure associated to the node: the node mobility is set to unity and the boolean variable SCHEDULE is asserted.

The changes in the node data structure affect the DG, but also affect the mobility of the predecessors/successors of the node just scheduled. To evaluate the changes a possibility is to run the previously defined ALAP/ASAP algorithm to obtain a new time frame for all the node. A more efficient solution is to run a recursive algorithm only on the children/parents of the scheduled node. This operation is performed by the NEW_ALAP/NEW_ASAP algorithms. The code for the NEW_ASAP algorithms is presented in fig. 5. Note that the recursion principle is the same used in the routines for the indirect force calculation. The only difference is that the ASAP control step of the node is now modified.

On the basis of the updated values, the new mobility of the successors/ancestors nodes is calculated. Note that at this point it may happen to find nodes with mobility equal to unity, e.g. scheduled nodes. This possibility is taken into account by the algorithm: the node is scheduled and the total number of unscheduled nodes is decreased of a unity.

The last step before running again the MIN_FORCE routine is to update the DG array with the UPD_DTABLE function. The top level routine (SCHEDULE) terminates when all the nodes of the DFG are annotated.

IV. EXPERIMENTAL RESULTS

Fodis has been carefully debugged with the Diffeq example analyzed in [1]. Since the number of operations and of control steps is very small, all the forces/distributions can be easily calculated and compared with the Fodis output log file. This example is just a part of the HAL benchmark which will be discussed in the next section.
Algorithm 5 UPDATE_ASAP routine.

UPDATE_ASAP (NODE_N, CSTEP_N)
FOR each children n_child
  IF n_child is not scheduled
    IF (CSTEP_N > CSTEP_n_child)
      tmp_cstep = CSTEP_n_child + 1
      set ASAP cstep of n_child = tmp_cstep
      UPDATE_ASAP (N_CHILD, TMP_CSTEP)
  END IF
END IF
END FOR
Return

Based on the Diffeq example a few remarks can be stated:
1) The final scheduling may depend from the order in which the graph is visited. Given the same force for two nodes, the move associated to the first visited node is scheduled.
2) For the same reason as before, the scheduling depends from the initial scheduling of the nodes. Reasonably, Fodis assumes that the nodes are initially scheduled in their ASAP control step.
3) If the number of control steps is bigger (but not too much) than the minimum feasible number of control steps, all the available control steps are used. This operation is done in order to minimize the concurrency of the same type of operation. This also implies that when the number of control steps exceed the total number of nodes, at the beginning of the scheduling Fodis will try to move the unscheduled operations in the last available control steps. When the number of control steps is too large a completely different behavior occurs (see section 4.2 for a detailed explanation).
4) Increasing the available scheduling control steps may lead to a worse global solution. During the execution Fodis may be trapped in a local minimum.

A. HAL benchmark

The Hal benchmark is an iterative solution of a second order differential equation. Fig. 1 shows the ASAP scheduling of the HAL DFG. Table I the max concurrencies of the Inp, Mul and Exp operations are reported for different scheduling. The other three types of operations, namely Sub, Add and Les are not considered in the table since their concurrency does not change between initial and final scheduling.

As clearly shown in fig. 1, at the first control step of the ASAP scheduling there is an extremely high concurrency of operation of type Inp. Running Fodis with the minimum number of control steps for HAL (c-steps=6) leads to a sensible improvement: the number of units executing the Inp operation is only 67% of the initial value; also, only 75% of the initial units executing the Mul operation and only half of the initial Exp units are now required. If the number of control step is increased of one (c-steps=7), a much better displacement of resources is achieved in the final scheduling.

B. Example 2: FIR filter

The FIR filter benchmark is used as an example because it has led to an improvement of Fodis. The ASAP scheduling of the FIR DFG is shown in fig. 2. As noticed the Inp nodes are all scheduled at the first control step. The FDS algorithm with the minimum number of control step plus one is expected to redistribute the Inp in an uniform way along the control time frame. The first version of Fodis was instead failing: the final scheduling was not as expected. The same kind of failure was also repeated running Fodis only on a small portion of the FIR filter graph (illustrated in fig. 3(a) ).

By choosing a total time frame of 5 control steps it is expected to lead to a uniform redistribution of the 4 concurrent Inp operations at control step 1. The best distribution would be achieved with 2 Inp operations shifted to the second control step. The final scheduling produced by Fodis is presented in fig. 3(b). As shown 3 Inp operation are still scheduled at the first control step; only one has been shifted.
At this point it is easy to think to errors in Fodis code, but this is not the case. In other words the solution found is a local minimum; it should be remarked at this point that FDS is an heuristic algorithm and the optimal solution is not guaranteed.

A closer look at the scheduling assignments sequence can fully explain the concept. The first node scheduled is obviously one of the 4 Inp node; Fodis schedules node number 10 at the second control step. At this point the entire right branch of the graph is scheduled (fig 3(b)), while the nodes of the left branch are still in their ASAP position and their mobility is 2. Which is the next node to be scheduled? Let’s look at the forces. Moving node 12 or 13 from their ASAP position to their ALAP position will lead to a positive force. In fact the direct force of the move is zero, since the Inp operation distribution values are the same in the first and second control steps. The indirect force would instead be a positive value because now the concurrency of both adders and multipliers is increased. The scheduling operation is then leaving the right branch node positions unchanged (e.g. in the initial ASAP position).

The behavior of the FDS algorithm can be improved with minor changes. The reason of the previous scheduling is rather simple: during the attempt to assign an operation to a different control step, the formula for the force calculation assumes static distribution values of the operation.

To improve the FDS behavior a different formula for the calculation of the force can be used [1]. The idea is to include the effect of the redistribution of the operation in the calculation of the force. The first step is the calculation of the new distribution value obtained by assigning the operation to the current control step. In the formula calculation the effect of the move is specified by assigning a weight of 2/3 to the ‘old’ distribution value and of 1/3 to the ‘new’ distribution value [1]. In this way the effects of the future assignment are taken into account in the force calculation. When the new formula is applied in the force calculation for the DFG of fig. 3, as expected, the two Inp operation are shifted to the second control step. On the basis of the results, Fodis was modified to use by default this improved method for force calculation.

In table II the results for the FIR benchmark are presented. The Exp operation is not included in the table since its concurrency is zero all the time. Fodis 1v indicates the old version of Fodis (e.g. no improved force calculation). When the minimum number of control steps (csteps=11) is used, both versions of Fodis lead to the same remarkable improvement of concurrency: 75% improvement is achieved for Inp, Add and Mul operations. Adding an extra control step (c-steps=12) leads to further improvements. The advantage of the improved formula can be noticed in the Inp operation concurrency values: the concurrency is further reduced of 50% against the initial 33% achieved with Fodis 1v.

Both in the case of the HAL benchmark and of the FIR benchmark increasing the number of csteps leads to suboptimal solutions. An increase of the concurrency is registered with respect to the scheduling with a smaller number of control steps. The heuristic nature of the FDS algorithm is

**TABLE II**

<table>
<thead>
<tr>
<th>Operation Concurrency in FIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inp</td>
</tr>
<tr>
<td>Initial ASAP scheduling</td>
</tr>
<tr>
<td>Fodis 1v with 11 cstep (min)</td>
</tr>
<tr>
<td>Fodis 1v with 12 cstep</td>
</tr>
<tr>
<td>Fodis with 11 cstep (min)</td>
</tr>
<tr>
<td>Fodis with 12 cstep</td>
</tr>
</tbody>
</table>
the explanation for such behavior: it should be noted that the probability of being trapped in a local minimum increases with the number of available scheduling control steps.

When the number of control steps is very large, the initial mobility of the nodes is wide. This implies that the DG of the operation contains almost uniformly distributed values. The following type of behavior can be observed:

- When the improved force calculation method is used, the final scheduling is concentrated in the first control steps leading to high concurrency. Consider what will happen in the Fir example when the schedule frame is very long: moving an Inp operation to a different step would lead to a positive force since the distribution probability of the operation in the destination cstep is now largely increased. The DF of the move is almost constant and, as expected, equal to 1/3. The same holds for the calculation of the Indirect Forces. For this reason, the second node scheduled by \textit{Fodis} is node 41 (fig. 2 and fig. 3); this set the minimum concurrency of the Inp units equal to 4.
- With the standard force calculation method, the scheduling will use all the available frame. As a result the first scheduled nodes will be placed at the bottom and the last scheduled nodes will be placed at the top, leaving a gap in the middle of the frame (e.g. empty csteps). This could be probably solved with an heuristic-based approach: e.g. in the evaluation of the moves for a specific node, if the force difference between two consecutive steps is smaller than a certain heuristic value then the node is assigned to the earliest control step. This simple solution will not affect the complexity and the execution time of the algorithm.

V. CONCLUSIONS

\textit{Fodis}, an implementation of the Force-Directed Scheduling algorithm, has been presented. The tool is able to generate schedulings which are in agreement with the FDS calculations. To decrease the probability of a local minimum trapping, forces are calculated in a “look ahead” fashion. Such implementation however does not guarantee that the final scheduling is an optimal one and it can actually lead to worse solution (with respect to the standard method) when the available schedule frame is long, as discussed in the previous section. Since the probability of being trapped in a local minimum increases with the number of control steps, further investigation could be done in a more efficient scheme for force calculation. A possible solution easily implementable would be to start the scheduling process without the “lookahead” scheme. When a certain number of nodes are scheduled, then the “lookahead” scheme can be used in the force calculation of the remaining unscheduled nodes. Another strategy could be assigning a different priority to the nodes to be scheduled, proportional to the number of operations of the same type not yet scheduled.

REFERENCES

[2] libHLS (v2.1.0) http://www.ece.neu.edu/groups/rpl/libHLS/